**Student ID:** Erik Iuhas 101076512

# HandyKeys Training and Validation Code

This notebook contains the models for training the HandyKeys Classifiers and testing

## Mount for Training

Training is done using my Google Drive. It will not be possible for you to test my training on your own google drive unless you set up the folder the same way as I did in at my drive link: [https://drive.google.com/drive/folders/1Tj5QfndOj4xlEvG-CxJihrJdXRqfIokj?usp=sharing](https://drive.google.com/drive/folders/1Tj5QfndOj4xlEvG-CxJihrJdXRqfIokj?usp=sharing)

```
from google.colab import drive
drive.mount('/content/drive')

folder_path = "drive/MyDrive/HandyKeys/Dataset"
```

```
Mounted at /content/drive
```

## Import Required Libraries (REQUIRED)

In this code block we are importing Tensorflow and Keras along with other libraries needed for training.

```
import os
import numpy as np
import cv2 as cv
import tensorflow as tf
import numpy as np
from PIL import Image
from itertools import cycle
from matplotlib import pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn import svm, datasets
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from scipy import interp
from sklearn.metrics import roc_auc_score
```

## Setup Training and Validation Paths

```
train_hand_1 = os.path.join(folder_path + '/train/Hand1')
train_hand_2 = os.path.join(folder_path + '/train/Hand2')
train_hand_3 = os.path.join(folder_path + '/train/Hand3')
train_hand_4 = os.path.join(folder_path + '/train/Hand4')
train_hand_5 = os.path.join(folder_path + '/train/Hand5')
train_hand_6 = os.path.join(folder_path + '/train/Hand6')
train_hand_7 = os.path.join(folder_path + '/train/Hand7')
train_none = os.path.join(folder_path + '/train/None')


validation_hand_1 = os.path.join(folder_path + '/validation/Hand1')
validation_hand_2 = os.path.join(folder_path + '/validation/Hand2')
validation_hand_3 = os.path.join(folder_path + '/validation/Hand3')
validation_hand_4 = os.path.join(folder_path + '/validation/Hand4')
validation_hand_5 = os.path.join(folder_path + '/validation/Hand5')
validation_hand_6 = os.path.join(folder_path + '/validation/Hand6')
validation_hand_7 = os.path.join(folder_path + '/validation/Hand7')
validation_none = os.path.join(folder_path + '/validation/None')
```

## Initialize Dataset and Validation Set

```
# set the size for the images being trained on.
image_size = 256

# apply data augmentation (Flip, and shift height and width) and regularize the photos to be within 0 to 1
train_datagen = ImageDataGenerator(rescale= 1./255,horizontal_flip = True,height_shift_range=0.1,width_shift_range=0.05,fill_mode='nearest')
validation_datagen = ImageDataGenerator(rescale= 1./255,horizontal_flip = True,height_shift_range=0.1,width_shift_range=0.05,fill_mode='nearest')

#select the batch size and initialize folders for training
train_generator = train_datagen.flow_from_directory(
        folder_path + '/train/',  # training image directory
        classes = ['Hand1', 'Hand2','Hand3', 'Hand4','Hand5', 'Hand6','Hand7', 'None'],
        target_size=(image_size, image_size),
        batch_size=22, # Because there are 792 images, we will use a batch size of 22
        class_mode= 'categorical', # using categorical because we have 8 classes
        shuffle=True)

# Flow validation images in batches of 19 using valid_datagen generator
validation_generator = validation_datagen.flow_from_directory(
        folder_path + '/validation/',  # training image directory
        classes = ['Hand1', 'Hand2','Hand3', 'Hand4','Hand5', 'Hand6','Hand7', 'None'],
        target_size=(image_size, image_size),
        batch_size=6, # Because there are 168 images, we will use a batch size of 6
        # Use binary labels
        class_mode= 'categorical', # using categorical because we have 8 classes
        shuffle=False)
```

```
Found 792 images belonging to 8 classes.
Found 168 images belonging to 8 classes.
```

## Create VGG Network Architecture

Following the architecture for Configuration A we are attempting to make a VGG network.

```
vgg_16_base = tf.keras.models.Sequential([

#First Layer, Convolution using the size of the image with depth 64
tf.keras.layers.Conv2D(64, (3,3), activation='relu',padding='same', input_shape=(image_size, image_size, 3)),
tf.keras.layers.MaxPooling2D(),

#Second Layer, Convolution with depth 128
tf.keras.layers.Conv2D(128, (3,3),padding='same', activation='relu'),
tf.keras.layers.MaxPooling2D(),

#Third Layer, Convolution with depth 256
```

```
tf.keras.layers.Conv2D(256, (3,3),padding='same', activation='relu'),
tf.keras.layers.Conv2D(256, (3,3),padding='same', activation='relu'),
tf.keras.layers.MaxPooling2D(),

#Fourth Layer, Convolution with depth 512
tf.keras.layers.Conv2D(512, (3,3),padding='same', activation='relu'),
tf.keras.layers.Conv2D(512, (3,3),padding='same', activation='relu'),
tf.keras.layers.MaxPooling2D(),

#Fifth Layer, Convolution with depth 512
tf.keras.layers.Conv2D(512, (3,3),padding='same', activation='relu'),
tf.keras.layers.Conv2D(512, (3,3),padding='same', activation='relu'),
tf.keras.layers.MaxPooling2D(),

tf.keras.layers.Flatten(),
tf.keras.layers.Dense(4096, activation='relu'),
tf.keras.layers.Dense(8, activation='softmax')])
```

## Display Network Architecture

```
vgg_16_base.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_16 (Conv2D)          (None, 256, 256, 64)      1792

 max_pooling2d_10 (MaxPoolin  (None, 128, 128, 64)     0
 g2D)

 conv2d_17 (Conv2D)          (None, 128, 128, 128)     73856

 max_pooling2d_11 (MaxPoolin  (None, 64, 64, 128)      0
 g2D)

 conv2d_18 (Conv2D)          (None, 64, 64, 256)       295168

 conv2d_19 (Conv2D)          (None, 64, 64, 256)       590080

 max_pooling2d_12 (MaxPoolin  (None, 32, 32, 256)      0
 g2D)

 conv2d_20 (Conv2D)          (None, 32, 32, 512)       1180160

 conv2d_21 (Conv2D)          (None, 32, 32, 512)       2359808

 max_pooling2d_13 (MaxPoolin  (None, 16, 16, 512)      0
 g2D)

 conv2d_22 (Conv2D)          (None, 16, 16, 512)       2359808

 conv2d_23 (Conv2D)          (None, 16, 16, 512)       2359808

 max_pooling2d_14 (MaxPoolin  (None, 8, 8, 512)        0
 g2D)

 flatten_2 (Flatten)         (None, 32768)             0

 dense_4 (Dense)             (None, 4096)              134221824

 dense_5 (Dense)             (None, 8)                 32776

=================================================================
Total params: 143,475,080
Trainable params: 143,475,080
Non-trainable params: 0
_________________________________________________________________
```

## Setup Checkpoints

```
checkpoint_path = "/tmp/cp-{epoch:04d}.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)

#Initialize checkpoints
cp_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_path,
    verbose=1,
    save_weights_only=True,
    save_freq=25) #save every batch

vgg_16_base.save_weights(checkpoint_path.format(epoch=0))
```

## Begin Training Classifier

```
vgg_16_base.compile(optimizer = tf.optimizers.Adam(),
              loss = 'binary_crossentropy',
              metrics=['accuracy',tf.keras.metrics.Recall()])
```

```
results = vgg_16_base.fit(train_generator,
      steps_per_epoch=36,  # because we have 792 training images and a batch size of 22 we need 36 steps per epoch
      epochs=35, # train for 30 epochs
      verbose=1,
      validation_data = validation_generator,
      callbacks = [cp_callback],
      validation_steps=28) # because we have 168 training images and a batch size of 6 we need 28 steps per epoch
```

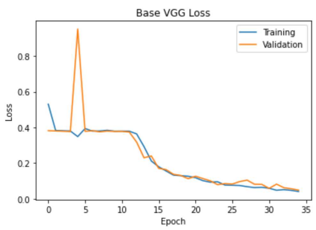
```
36/36 [==============================] - 26s 729ms/step - loss: 0.0933 - accuracy: 0.8321 - recall_2: 0.8018 - val_loss: 0.1008 - val_accuracy: 0.8274 - val_recall_2: 0.7857
Epoch 24/35
21/36 [================>.............] - ETA: 4s - loss: 0.0957 - accuracy: 0.8442 - recall_2: 0.8052
Epoch 24: saving model to /tmp/cp-0024.ckpt
36/36 [==============================] - 20s 554ms/step - loss: 0.0953 - accuracy: 0.8346 - recall_2: 0.8030 - val_loss: 0.0794 - val_accuracy: 0.8869 - val_recall_2: 0.8274
Epoch 25/35
10/36 [=======>......................] - ETA: 8s - loss: 0.0796 - accuracy: 0.8545 - recall_2: 0.8318
Epoch 25: saving model to /tmp/cp-0025.ckpt
35/36 [============================>.] - ETA: 0s - loss: 0.0759 - accuracy: 0.8675 - recall_2: 0.8468
Epoch 25: saving model to /tmp/cp-0025.ckpt
36/36 [==============================] - 26s 734ms/step - loss: 0.0770 - accuracy: 0.8649 - recall_2: 0.8447 - val_loss: 0.0851 - val_accuracy: 0.8690 - val_recall_2: 0.8333
Epoch 26/35
24/36 [===================>..........] - ETA: 3s - loss: 0.0749 - accuracy: 0.8617 - recall_2: 0.8409
Epoch 26: saving model to /tmp/cp-0026.ckpt
36/36 [==============================] - 20s 563ms/step - loss: 0.0760 - accuracy: 0.8687 - recall_2: 0.8485 - val_loss: 0.0826 - val_accuracy: 0.8690 - val_recall_2: 0.8333
Epoch 27/35
13/36 [=========>....................] - ETA: 7s - loss: 0.0661 - accuracy: 0.9056 - recall_2: 0.8951
Epoch 27: saving model to /tmp/cp-0027.ckpt
36/36 [==============================] - 20s 552ms/step - loss: 0.0747 - accuracy: 0.8838 - recall_2: 0.8624 - val_loss: 0.0966 - val_accuracy: 0.8214 - val_recall_2: 0.7917
Epoch 28/35
 2/36 [>.............................] - ETA: 10s - loss: 0.0754 - accuracy: 0.8864 - recall_2: 0.8636
Epoch 28: saving model to /tmp/cp-0028.ckpt
27/36 [=====================>........] - ETA: 5s - loss: 0.0613 - accuracy: 0.9024 - recall_2: 0.8788
Epoch 28: saving model to /tmp/cp-0028.ckpt
36/36 [==============================] - 26s 741ms/step - loss: 0.0680 - accuracy: 0.8902 - recall_2: 0.8674 - val_loss: 0.1047 - val_accuracy: 0.8155 - val_recall_2: 0.7857
Epoch 29/35
```

```
Epoch 29/35
16/36 [============>.................] - ETA: 6s - loss: 0.0699 - accuracy: 0.8835 - recall_2: 0.8807
Epoch 29: saving model to /tmp/cp-0029.ckpt
36/36 [==============================] - 20s 559ms/step - loss: 0.0630 - accuracy: 0.8990 - recall_2: 0.8876 - val_loss: 0.0816 - val_accuracy: 0.8750 - val_recall_2: 0.8571
Epoch 30/35
 5/36 [===>..........................] - ETA: 10s - loss: 0.0626 - accuracy: 0.8818 - recall_2: 0.8818
Epoch 30: saving model to /tmp/cp-0030.ckpt
30/36 [========================>.....] - ETA: 3s - loss: 0.0606 - accuracy: 0.9000 - recall_2: 0.8848
Epoch 30: saving model to /tmp/cp-0030.ckpt
36/36 [==============================] - 26s 729ms/step - loss: 0.0645 - accuracy: 0.8990 - recall_2: 0.8851 - val_loss: 0.0807 - val_accuracy: 0.8869 - val_recall_2: 0.8512
Epoch 31/35
19/36 [==============>...............] - ETA: 5s - loss: 0.0647 - accuracy: 0.8995 - recall_2: 0.8756
Epoch 31: saving model to /tmp/cp-0031.ckpt
36/36 [==============================] - 21s 582ms/step - loss: 0.0592 - accuracy: 0.9091 - recall_2: 0.8864 - val_loss: 0.0583 - val_accuracy: 0.9107 - val_recall_2: 0.8869
Epoch 32/35
 8/36 [=====>........................] - ETA: 8s - loss: 0.0437 - accuracy: 0.9375 - recall_2: 0.9261
Epoch 32: saving model to /tmp/cp-0032.ckpt
33/36 [==========================>...] - ETA: 1s - loss: 0.0451 - accuracy: 0.9311 - recall_2: 0.9256
Epoch 32: saving model to /tmp/cp-0032.ckpt
36/36 [==============================] - 26s 721ms/step - loss: 0.0480 - accuracy: 0.9242 - recall_2: 0.9192 - val_loss: 0.0822 - val_accuracy: 0.8750 - val_recall_2: 0.8571
Epoch 33/35
22/36 [=================>............] - ETA: 4s - loss: 0.0521 - accuracy: 0.9174 - recall_2: 0.9070
Epoch 33: saving model to /tmp/cp-0033.ckpt
36/36 [==============================] - 20s 558ms/step - loss: 0.0514 - accuracy: 0.9179 - recall_2: 0.9053 - val_loss: 0.0621 - val_accuracy: 0.9107 - val_recall_2: 0.8869
Epoch 34/35

11/36 [========>.....................] - ETA: 8s - loss: 0.0416 - accuracy: 0.9421 - recall_2: 0.9339
Epoch 34: saving model to /tmp/cp-0034.ckpt
36/36 [==============================] - 20s 564ms/step - loss: 0.0474 - accuracy: 0.9230 - recall_2: 0.9116 - val_loss: 0.0564 - val_accuracy: 0.9226 - val_recall_2: 0.9048
Epoch 35/35

Epoch 35: saving model to /tmp/cp-0035.ckpt
25/36 [===================>..........] - ETA: 3s - loss: 0.0324 - accuracy: 0.9545 - recall_2: 0.9309
Epoch 35: saving model to /tmp/cp-0035.ckpt
```

## Plot Loss

Plotting the loss to determine when to stop training the network.

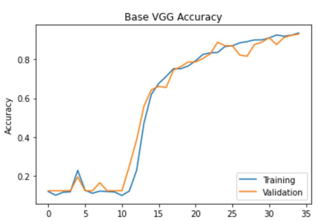
```
#Plot the loss
plt.plot(results.history['loss']) #Testing
plt.plot(results.history['val_loss']) #validation
#labels
plt.title('Base VGG Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```




## Plot Accuracy

Plotting the accuracy to determine when the network performed best.

```
#Plot the loss
plt.plot(results.history['accuracy']) #Testing
plt.plot(results.history['val_accuracy']) #Validation
#Labels
plt.title('Base VGG Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='lower right')
plt.show()
```




## Save Weights

Save VGG weights

```
vgg_16_base.save_weights(folder_path + "/vgg_base.h5")
```

## Create new Convolutional Neural Network

Due to the higher variance in perforamcne for validatiopn accuracy i am looking to make a more shallow CNN network including dropout layers to improve the performance and increase regularization. I am looking to do this because the application will be running on a local computer and if the model is too large it won't be able to make inferences fast enough to be considered useful.

## New Network - TinyHand

The machine learning network that I will construct will be called TinyHand going further. This is because it will be a smaller CNN and Hand comes from the Project name "HandyKeys"

```
tiny_hand = tf.keras.models.Sequential([

#First Layer, First Layer remains the same
tf.keras.layers.Conv2D(64, (3,3), activation='relu',padding='same', input_shape=(image_size, image_size, 3)),
```

```
tf.keras.layers.MaxPooling2D(),
tf.keras.layers.Dropout(0.2), #Indroducing Dropout Layers which improves regularization

#Second Layer, Convolution with depth 32
tf.keras.layers.Conv2D(32, (3,3),padding='same', activation='relu'),
tf.keras.layers.Conv2D(32, (3,3),padding='same', activation='relu'),
tf.keras.layers.MaxPooling2D(),
tf.keras.layers.Dropout(0.2), #Indroducing Dropout Layers which improves regularization

#Third Layer, Convolution with depth 64
tf.keras.layers.Conv2D(64, (3,3),padding='same', activation='relu'),
tf.keras.layers.Conv2D(64, (3,3),padding='same', activation='relu'),
tf.keras.layers.MaxPooling2D(),
tf.keras.layers.Dropout(0.2), #Indroducing Dropout Layers which improves regularization

#Fourth Layer, Convolution with depth 128 Increasing the count of Conv to 3
tf.keras.layers.Conv2D(128, (3,3),padding='same', activation='relu'),
tf.keras.layers.Conv2D(128, (3,3),padding='same', activation='relu'),
tf.keras.layers.Conv2D(128, (3,3),padding='same', activation='relu'),
tf.keras.layers.MaxPooling2D(),
tf.keras.layers.Dropout(0.2), #Indroducing Dropout Layers which improves regularization

#Fifth Layer, Convolution with depth 256
tf.keras.layers.Conv2D(256, (3,3),padding='same', activation='relu'),
tf.keras.layers.Conv2D(256, (3,3),padding='same', activation='relu'),
tf.keras.layers.MaxPooling2D(),

tf.keras.layers.Flatten(),
tf.keras.layers.Dense(2048, activation='relu'),
tf.keras.layers.Dense(8, activation='softmax')])
```

## TinyHand Summary

As seen in the summary we've reduced the number of parameters by

```
tiny_hand.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_24 (Conv2D)          (None, 256, 256, 64)      1792

 max_pooling2d_15 (MaxPoolin  (None, 128, 128, 64)     0
 g2D)

 dropout (Dropout)           (None, 128, 128, 64)      0

 conv2d_25 (Conv2D)          (None, 128, 128, 32)      18464

 conv2d_26 (Conv2D)          (None, 128, 128, 32)      9248

 max_pooling2d_16 (MaxPoolin  (None, 64, 64, 32)       0
 g2D)

 dropout_1 (Dropout)         (None, 64, 64, 32)        0

 conv2d_27 (Conv2D)          (None, 64, 64, 64)        18496

 conv2d_28 (Conv2D)          (None, 64, 64, 64)        36928

 max_pooling2d_17 (MaxPoolin  (None, 32, 32, 64)       0
 g2D)

 dropout_2 (Dropout)         (None, 32, 32, 64)        0

 conv2d_29 (Conv2D)          (None, 32, 32, 128)       73856

 conv2d_30 (Conv2D)          (None, 32, 32, 128)       147584

 conv2d_31 (Conv2D)          (None, 32, 32, 128)       147584

 max_pooling2d_18 (MaxPoolin  (None, 16, 16, 128)      0
 g2D)

 dropout_3 (Dropout)         (None, 16, 16, 128)       0

 conv2d_32 (Conv2D)          (None, 16, 16, 256)       295168

 conv2d_33 (Conv2D)          (None, 16, 16, 256)       590080

 max_pooling2d_19 (MaxPoolin  (None, 8, 8, 256)        0
 g2D)

 flatten_3 (Flatten)         (None, 16384)             0

 dense_6 (Dense)             (None, 2048)              33556480

 dense_7 (Dense)             (None, 8)                 16392

=================================================================
Total params: 34,912,072
Trainable params: 34,912,072
Non-trainable params: 0
_________________________________________________________________
```

## Prepare Checkpoints

```
checkpoint_path = "/tmp/cp-{epoch:04d}.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)

#Initialize variables for checkpoint testing
cp_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_path,
    verbose=1,
    save_weights_only=True,
    save_freq=25) #save every batch

tiny_hand.save_weights(checkpoint_path.format(epoch=0))
```

## Begin Training TinyHand

Because TinyHand is a smaller network it will likely take more time for it to reach an optimal loss, to account for this the epohcs were increased to 50 improve learning. This was also found out by earlier training where the loss was still occuring during the original length of 35.

```
tiny_hand.compile(optimizer = tf.optimizers.Adam(),
              loss = 'binary_crossentropy',
              metrics=['accuracy',tf.keras.metrics.Recall()])
```

```
results_tiny = tiny_hand.fit(train_generator,
```

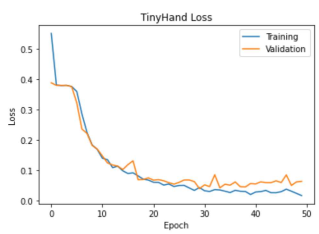
```
    steps_per_epoch=36,  # because we have 792 training images and a batch size of 22 we need 36 steps per epoch
    epochs=50, # train for 50 epochs
    verbose=1,
    validation_data = validation_generator,
    callbacks = [cp_callback],
    validation_steps=28) # because we have 168 training images and a batch size of 6 we need 28 steps per epoch
```

```
Epoch 21: saving model to /tmp/cp-0021.ckpt
29/36 [=======================>......] - ETA: 2s - loss: 0.0645 - accuracy: 0.8887 - recall_3: 0.8746
Epoch 21: saving model to /tmp/cp-0021.ckpt
36/36 [==============================] - 16s 434ms/step - loss: 0.0613 - accuracy: 0.8977 - recall_3: 0.8838 - val_loss: 0.0682 - val_accuracy: 0.9048 - val_recall_3: 0.8750
Epoch 22/50
18/36 [==============>...............] - ETA: 5s - loss: 0.0576 - accuracy: 0.9066 - recall_3: 0.8990
Epoch 22: saving model to /tmp/cp-0022.ckpt
36/36 [==============================] - 15s 409ms/step - loss: 0.0608 - accuracy: 0.9015 - recall_3: 0.8851 - val_loss: 0.0699 - val_accuracy: 0.8988 - val_recall_3: 0.8690
Epoch 23/50
 7/36 [====>.........................] - ETA: 9s - loss: 0.0604 - accuracy: 0.9026 - recall_3: 0.8896
Epoch 23: saving model to /tmp/cp-0023.ckpt
32/36 [=========================>....] - ETA: 1s - loss: 0.0503 - accuracy: 0.9247 - recall_3: 0.9062
Epoch 23: saving model to /tmp/cp-0023.ckpt
36/36 [==============================] - 16s 438ms/step - loss: 0.0519 - accuracy: 0.9217 - recall_3: 0.9015 - val_loss: 0.0662 - val_accuracy: 0.9048 - val_recall_3: 0.8750
Epoch 24/50
21/36 [================>.............] - ETA: 4s - loss: 0.0499 - accuracy: 0.9286 - recall_3: 0.9134
Epoch 24: saving model to /tmp/cp-0024.ckpt
36/36 [==============================] - 15s 409ms/step - loss: 0.0554 - accuracy: 0.9154 - recall_3: 0.8990 - val_loss: 0.0599 - val_accuracy: 0.9226 - val_recall_3: 0.8929
Epoch 25/50
10/36 [=======>......................] - ETA: 8s - loss: 0.0416 - accuracy: 0.9500 - recall_3: 0.9273
Epoch 25: saving model to /tmp/cp-0025.ckpt
35/36 [============================>.] - ETA: 0s - loss: 0.0478 - accuracy: 0.9351 - recall_3: 0.9208
Epoch 25: saving model to /tmp/cp-0025.ckpt
36/36 [==============================] - 16s 444ms/step - loss: 0.0475 - accuracy: 0.9356 - recall_3: 0.9217 - val_loss: 0.0550 - val_accuracy: 0.9048 - val_recall_3: 0.8869
Epoch 26/50
24/36 [===================>..........] - ETA: 3s - loss: 0.0519 - accuracy: 0.9318 - recall_3: 0.9223
Epoch 26: saving model to /tmp/cp-0026.ckpt
36/36 [==============================] - 16s 436ms/step - loss: 0.0506 - accuracy: 0.9331 - recall_3: 0.9230 - val_loss: 0.0607 - val_accuracy: 0.8810 - val_recall_3: 0.8750
Epoch 27/50
13/36 [=========>....................] - ETA: 7s - loss: 0.0379 - accuracy: 0.9476 - recall_3: 0.9406
Epoch 27: saving model to /tmp/cp-0027.ckpt
36/36 [==============================] - 15s 409ms/step - loss: 0.0510 - accuracy: 0.9306 - recall_3: 0.9230 - val_loss: 0.0688 - val_accuracy: 0.8929 - val_recall_3: 0.8631
Epoch 28/50
 2/36 [>.............................] - ETA: 11s - loss: 0.0627 - accuracy: 0.9318 - recall_3: 0.9091
Epoch 28: saving model to /tmp/cp-0028.ckpt
27/36 [=====================>........] - ETA: 3s - loss: 0.0455 - accuracy: 0.9310 - recall_3: 0.9141
Epoch 28: saving model to /tmp/cp-0028.ckpt
36/36 [==============================] - 15s 430ms/step - loss: 0.0426 - accuracy: 0.9331 - recall_3: 0.9192 - val_loss: 0.0687 - val_accuracy: 0.8929 - val_recall_3: 0.8929
Epoch 29/50
16/36 [============>.................] - ETA: 6s - loss: 0.0398 - accuracy: 0.9432 - recall_3: 0.9290
Epoch 29: saving model to /tmp/cp-0029.ckpt
36/36 [==============================] - 15s 408ms/step - loss: 0.0345 - accuracy: 0.9470 - recall_3: 0.9407 - val_loss: 0.0635 - val_accuracy: 0.9226 - val_recall_3: 0.9167
Epoch 30/50
 5/36 [===>..........................] - ETA: 12s - loss: 0.0475 - accuracy: 0.9364 - recall_3: 0.9364
Epoch 30: saving model to /tmp/cp-0030.ckpt
30/36 [========================>.....] - ETA: 2s - loss: 0.0449 - accuracy: 0.9485 - recall_3: 0.9394
Epoch 30: saving model to /tmp/cp-0030.ckpt
36/36 [==============================] - 16s 443ms/step - loss: 0.0438 - accuracy: 0.9495 - recall_3: 0.9369 - val_loss: 0.0403 - val_accuracy: 0.9345 - val_recall_3: 0.9345
Epoch 31/50
19/36 [==============>...............] - ETA: 5s - loss: 0.0351 - accuracy: 0.9474 - recall_3: 0.9354
Epoch 31: saving model to /tmp/cp-0031.ckpt
36/36 [==============================] - 15s 410ms/step - loss: 0.0333 - accuracy: 0.9508 - recall_3: 0.9394 - val_loss: 0.0527 - val_accuracy: 0.9226 - val_recall_3: 0.9167
Epoch 32/50
 8/36 [=====>........................] - ETA: 8s - loss: 0.0190 - accuracy: 0.9716 - recall_3: 0.9602
Epoch 32: saving model to /tmp/cp-0032.ckpt
33/36 [==========================>...] - ETA: 1s - loss: 0.0312 - accuracy: 0.9504 - recall_3: 0.9449
Epoch 32: saving model to /tmp/cp-0032.ckpt
36/36 [==============================] - 15s 429ms/step - loss: 0.0309 - accuracy: 0.9520 - recall_3: 0.9470 - val_loss: 0.0471 - val_accuracy: 0.9524 - val_recall_3: 0.9524
```

## Plot Loss

Plotting the loss to determine when to stop training the network.

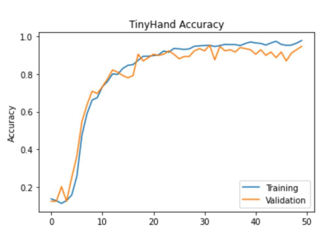
```
#Plot the loss
plt.plot(results_tiny.history['loss']) #Testing Loss
plt.plot(results_tiny.history['val_loss']) #Validation Loss

#Labels
plt.title('TinyHand Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```




## Plot Accuracy

Plotting the accuracy to determine when the network performed best.

```
#Plot the accuracy
plt.plot(results_tiny.history['accuracy']) #Testing Accuracy
plt.plot(results_tiny.history['val_accuracy']) #Validation Accuracy
#Add Labels
plt.title('TinyHand Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='lower right')
plt.show()
```



```
tiny_hand.save_weights(folder_path + "/tiny_hand_final.h5")
```

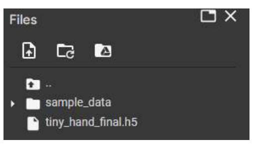
## Build TinyHand Architecture (REQUIRED)

Repeating the tiny_hand structure above. You need to run this section in order to load the weights from the "tiny_hand_final.h5" weights.

```
image_size = 256
tiny_hand = tf.keras.models.Sequential([

#First Layer, First Layer remains the same
tf.keras.layers.Conv2D(64, (3,3), activation='relu',padding='same', input_shape=(image_size, image_size, 3)),
tf.keras.layers.MaxPooling2D(),
tf.keras.layers.Dropout(0.2), #Indroducing Dropout Layers which improves regularization

#Second Layer, Convolution with depth 32
tf.keras.layers.Conv2D(32, (3,3),padding='same', activation='relu'),
tf.keras.layers.Conv2D(32, (3,3),padding='same', activation='relu'),
tf.keras.layers.MaxPooling2D(),
tf.keras.layers.Dropout(0.2), #Indroducing Dropout Layers which improves regularization

#Third Layer, Convolution with depth 64
tf.keras.layers.Conv2D(64, (3,3),padding='same', activation='relu'),
tf.keras.layers.Conv2D(64, (3,3),padding='same', activation='relu'),
tf.keras.layers.MaxPooling2D(),
tf.keras.layers.Dropout(0.2), #Indroducing Dropout Layers which improves regularization

#Fourth Layer, Convolution with depth 128 Increasing the count of Conv to 3
tf.keras.layers.Conv2D(128, (3,3),padding='same', activation='relu'),
tf.keras.layers.Conv2D(128, (3,3),padding='same', activation='relu'),
tf.keras.layers.Conv2D(128, (3,3),padding='same', activation='relu'),
tf.keras.layers.MaxPooling2D(),
tf.keras.layers.Dropout(0.2), #Indroducing Dropout Layers which improves regularization

#Fifth Layer, Convolution with depth 256
tf.keras.layers.Conv2D(256, (3,3),padding='same', activation='relu'),
tf.keras.layers.Conv2D(256, (3,3),padding='same', activation='relu'),
tf.keras.layers.MaxPooling2D(),

tf.keras.layers.Flatten(),
tf.keras.layers.Dense(2048, activation='relu'),
tf.keras.layers.Dense(8, activation='softmax')])
```

## Load the Weight File (REQUIRED)

Ensure you upload the file into the base file directory located at the left. It should look like the image below after uploading the file.




Use the weight file included in my submission in the HandyKeysApp Directory.

You can also find the weights in my Dataset Folder at this Drive link: [https://drive.google.com/drive/folders/1Tj5QfndOj4xlEvG-CxJihrJdXRqflokj?usp=sharing](https://drive.google.com/drive/folders/1Tj5QfndOj4xlEvG-CxJihrJdXRqflokj?usp=sharing)

```
tiny_hand.load_weights("tiny_hand_final.h5")
```

## Test TinyHand (REQUIRED)

This section of the Notebook allows for you to load the model of tiny hand and the validation set that was being used in the training to check that my network is working. By using the small validation set I provided you with you can pick from the examples to upload one at a time to receive the classification and the photos or just upload all at once to recieve the classifications only.

```
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predict hand gestures
  path = '/content/' + fn
  img = image.load_img(path, target_size=(256, 256))
  array_img = image.img_to_array(img)
  plt.imshow(array_img/255.)
  array_img = np.expand_dims(array_img, axis=0)
  images = np.vstack([array_img])
  classes = tiny_hand.predict(images, batch_size=1)
  # print hot-ones encodeing
  print(classes)

  #select largest index
  max_index = np.argmax(classes[0])

  # Print the hand key number.
  if max_index == 0:
    print("Hand is Key 1")
  elif max_index == 1:
    print("Hand is Key 2")
  elif max_index == 2:
    print("Hand is Key 3")
  elif max_index == 3:
    print("Hand is Key 4")
  elif max_index == 4:
    print("Hand is Key 5")
  elif max_index == 5:
    print("Hand is Key 6")
  elif max_index == 6:
    print("Hand is Key 7")
  else:
    print("No Hand. Nothing.")
```

Choose Files No file chosen